DOCUMENT RESUME

ED 428 736 IR 019 397

AUTHOR	Westhoff, Dirk; Unger, Claus
TITLE	"Campus" - An Agent-Based Platform for Distance Education.
PUB DATE	1998-06-00
NOTE	7p.; In: ED-MEDIA/ED-TELECOM 98 World Conference on Educational Multimedia and Hypermedia & World Conference on Educational Telecommunications. Proceedings (10th, Freiburg, Germany, June 20-25, 1998); see IR 019 307. Figures may not reproduce clearly.
PUB TYPE	Reports - Descriptive (141) -- Speeches/Meeting Papers (150)
EDRS PRICE	MF01/PC01 Plus Postage.
DESCRIPTORS	*Access to Information; *Artificial Intelligence; *Computer Interfaces; Computer Mediated Communication; Computer Oriented Programs; *Computer System Design; Computer Uses in Education; Distance Education; Educational Technology; Foreign Countries; Higher Education; Information Retrieval; *Internet; Robotics
IDENTIFIERS	Java Programming Language

ABSTRACT

This paper presents "Campus," an environment that allows University of Hagen (Germany) students to connect briefly to the Internet but remain represented by personalized, autonomous agents that can fulfill a variety of information, communication, planning, and cooperation tasks. A brief survey is presented of existing mobile agent system environments, all of which are based on a central architecture requiring one or more servers to be permanently active and reachable. The Agent Application Programming Interface (AAPI) package is introduced; AAPI is an extension of the Java Class Hierarchy that supports the design and implementation of systems of mobile, autonomous agents and is based upon decentralized control structures. Derived from the AAPI package, "Campus" offers a variety of "Campus Intercommunication Agents" that can perform the following functions on behalf of their owners: retrieve information from libraries, search machines, and faculty/registrar blackboards; exchange information with other agents; search for individual agents; cooperate with other agents in setting up individual working groups; enroll their owners into existing working groups; and arrange meetings between owners. A table presents properties of mobile agent systems. Four figures illustrate migration of an AAPI agent, reverse routing, the two-layered network of "Campus," and the agents' docking and route windows. (DLS)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED 428 736

"Campus" - an Agent-based Platform for Distance Education

Dirk Westhoff, Claus Unger
University of Hagen, Department for Computer Science
58084 Hagen, Germany
dirk.westhoff@fernuni-hagen.de, claus.unger@fernuni-hagen.de

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
☐ This document has been reproduced as received from the person or organization originating it.
☐ Minor changes have been made to improve reproduction quality.
• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
G.H. Marks
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

Abstract: *Campus* supports the communication and co-operation between a distance teaching university and its students in an Internet environment. While students usually connect quite rarely to the Internet, they can continuously be represented in the Internet by their individual autonomous agents which can fulfil a variety of tasks for their owners. *Campus* is being implemented with the help of the Agent Application Programming Interface (AAPI) package. AAPI is an extension of the Java Class Hierarchy and has been developed at the University of Hagen; it supports the design and implementation of systems of mobile, autonomous agents and is based upon decentralised control structures. Derived from the AAPI package, Campus offers a variety of *Campus Intercommunication Agents*.

1 Introduction

The University of Hagen is a distance teaching university with about 55.000 students, most of them spread all over Germany, some of them even over the whole world. Most of these students have access to the Internet, but because of connection costs are linked to the Internet just for short intervals. *Campus* provides an environment in which students are continuously represented by their own personalised, autonomous agents which can fulfil a variety of information, communication, planning and co-operation tasks on behalf of their owners. Each student runs on her computer a *Campus* environment, populated with several Campus Intercommunication Agents (*CIAgents*). The student charges her agent with a list of tasks, briefly connects to the Internet, releases the agent to the network, and disconnects again. According to their tasks, the agents migrate through the network, collect information for their owners, communicate and co-operate with other agents, until they are finally picked up again by their owners as soon as they reconnect to the network. Thus, while students are just rarely connected to the network, their agents can continuously represent them in the network. Beside minimising the student access times to the network, co-operation and communication between agents may stimulate co-operation and communication between students as well. *Campus* and its services are dealt with in more detail in chapter 4. Compared with the traditional server/client paradigm, mobile, autonomous agents in general show several essential advantages: transporting the algorithms to the data and substituting global network-wide communication and co-operation by local communication and co-operation may significantly reduce communication costs and the amount of network resources needed. On the other hand, security, authentication and accounting problems have to be solved for agent systems. *Campus* is being implemented as an agent application on top of the Agent Application Programming Interface (AAPI) package.
In the following chapters, we briefly survey existing mobile agent systems, describe the basic concepts of the AAPI package, introduce *Campus* in some more detail, and conclude with some ideas on future work.

2 Survey of Mobile Agent-Systems

This chapter gives a brief survey of some existing mobile agent system environments. The java-based *Aglet* workbench of IBM [IBM 96] is an extension of mobile java applets. Similar to an applet, an aglet's binary code migrates through the network, but in contrast to an applet, the state of an aglet is transported together with the aglet. States are defined in terms of creation, migration, activation, deactivation and termination events. For each state, the aglet workbench offers the aglet programmer adequate sets of methods. Global and local communication between aglets is realised via the communication mechanisms *Remote Procedure Call* (RPC) and *message passing*. In both cases, aglets have to create so-called *Aglet-Proxies*. These are the aglets' communication stations which also protect aglets from unauthorised manipulation.
Agent Tcl [Kotz et al. 96],[Gray et al. 96] is a mobile agent-system which was developed at the Dartmouth College.

IR019397

BEST COPY AVAILABLE

Agents are realised in Agent Tcl, an extension of the scripting language *Tool command language* (Tcl). For communication between agents, Agent Tcl uses the *Agent Remote Procedure Call* (ARPC) or a paging mechanism. To work under non-permanent network connections, Agent Tcl contains the *Laptop Docking System*, which assigns every Laptop a permanent docking computer. When the Laptop is not connected to the network, the docking computer serves as target for the agent. Security services, based on Safe Tcl, protect a computer from malicious agents. Security services to protect agents against malicious environments have not been realised yet.
Mole [Straßer et al. 96],[Hohl 95] is the prototype of an agent system. Stationary *System-Agents* manage the resources and services of one place (set of computers). If mobile *User Agents* visit these places, the *System Agent* assigns special resources or services to these *User Agents*. Mole supports local and global communication [Röhrle et al. 96] with RPC and message passing mechanisms. If an agent wants to communicate globally, it sends its message to a central *Mailbox Agent*, which forwards the message to the other partner.
JAE (Java Agent Environment) [Park et al. 97] has been drafted at the technical University Aachen. JAE focusses on the integration of wireless (e.g. mobile phone) and Internet agent technologies. JAE introduces the concept for Personal Digital Assistants (PDA). JAE distinguishes between *agent servers*, *mobile agents* and *stationary service agents*. A computer based agent server provides an environment for incoming mobile agents, and is supported by service agents. Only mobile agents can leave a computer and travel to another computer. Additional approaches can be found in [Li et al. 96].

All the described agent systems are based upon a central architecture, i.e. one or more servers have to be permanently active and reachable. To avoid bottlenecks and problems with server failures, as well as to gain general experiences with decentralised architectures, the AAPI package is, following the basic Internet paradigm, uncompromisingly based upon a decentralised approach. [Tab.1] compares the described systems and the AAPI package approach:

	IBM Aglets	*Agent Tcl*	*Mole*	*JAE*	*AAPI*
agent migration	+	+	+	+	+
state-of-life-API	+				+
local communication	+	+	+	+	+
global communication	+	+	(+)	+	+
security for the agent	+				
security for the computer		+		+	(+)
non-permament connections	+	+		(+)	+
decentral IP-management					+
authentification					(+)
accounting					(+)
platform independency	+		+	+	+
					(+) := (planned) quality of the agent-system

Table 1: *Properties of mobile agent systems*

3 The AAPI Package

The runtime environment for an agent is called an active *context*. Except when migrating between contexts, agents are always embedded in a context. A context can:

- *start and ship its own agents;*
- *dock (i.e. receive, start and ship) foreign, travelling agents;*
- *synchronise several running agents;*
- *communicate with its own travelling agents;*
- *support communication with foreign agents in foreign contexts;*

The agent's statical *profile* is defined by a set of *attributes*, e.g. identifier, creation date, owner, task description, etc.
When the context starts one of its own agents, it defines an initial *route* for the agent's tour through the Internet. Such a route may be a simple list or, e.g., a complex path graph of Internet addresses. During its journey, the agent can extend its initial route.
If the agent wants to move from one context to another, the binary-code of the agent has to be sent together with its profile and route.
Agent, profile, route and binary code are represented as objects which are linked to each other. At the stage of migration, they change their representations into data-streams and, via its Internet address, move to the next site on the agent's route [Fig.1].

Figure 1: *Migration of an AAPI agent*

Agents working in the same context are represented as parallel threads. They can communicate and co-operate with each other via *ComObjects*. When an agent wants to start a communication with a parallel agent, it creates a ComObject for this agent, which beside the message may contain algorithms in form of binary code to be executed by the other agent. It may also contain structured containers for results. Within its thread, each agent is continuously looking for ComObjects another agent has created for it. Communication between agents across different contexts in different computers works in a similar way. The agent creates a ComObject in its context and sends this object to the context of the target agent, which in turn executes the requested tasks and sends the results back.

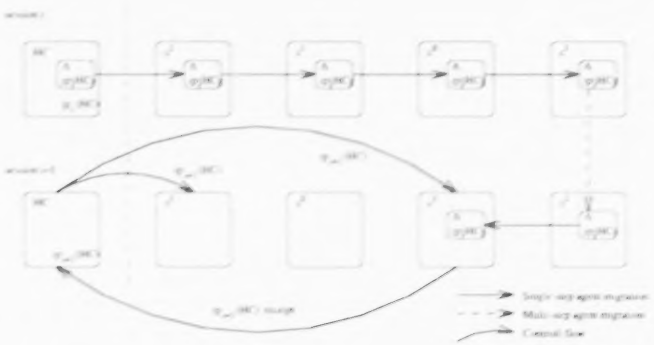

Figure 2: *Reverse routing*

The Internet address of its owning context always defines the last station in an agent's route. As mentioned before, our students' computers are quite rarely linked to the network and may get assigned different Internet addresses for different sessions. In such a case an agent cannot find home and present its results to its owner. The following algorithm solves this problem [Fig.2]: whenever the home context (HC) of an agent changes its Internet address from $ip_i(HC)$ to $ip_{i+1}(HC)$, it immediately informs all contexts c^j on the agent's reverse initial route until it finds the context where the agent is actually working in. This context changes the target station in the agent's route

BEST COPY AVAILABLE

accordingly and confirms the successful change to the agent's home context.
If the agent has extended its initial route and is actually working in a context c^E that is part of the route extension, the home context can't inform the agent about a changed internet adress. In this case, after the agent has finished working on the extended contexts, the agent migrates to an active context of its initial route, to wait for information of its home context. Here the agent can be reached again, to change the target station in its agent's route.

This reverse routing algorithm assumes that the home context can reach its agent. There are several reasons why a home context may not be able to reach its agent A:

- *unavailability of A during the migration;*
- *regular termination of A's actual working context;*
- *irregular termination of A's actual working context;*
- *interrupted connection to A's actual working context;*
- *Agent A is working in a context c^E;*

While the first case can be solved by handling the migration of an agent as an atomic operation [Mira da Silva et al. 97],[Klar 96],[Westhoff 97], the second one can be handled by enforcing contexts not to close as long as they contain active agents. The last three cases can only be handled by restarting the reverse routing algorithm after a certain time-out.

The above algorithm assumes that only the home computer of an agent may get disconnected from the network and change its Internet address. We are presently working on an extension of the above algorithm which allows arbitrary computers to change their Internet addresses and to inform all interested computers in the network. To strictly follow the agent paradigm, even short messages between contexts are modelled as agents.
The classes and methods of the AAPI package realise a basic mobile agent, i.e. they support an agent's creation, travelling, working, self-copying and termination. More advanced agents can easily be defined within the AAPI environment by simply overwriting the corresponding methods. Details on the AAPI package and how to use it can be found in [Westhoff 97].

4 Campus

While lecturers as well as all university institutions are almost permanently connected to the Internet with constant Internet addresses, most of the students are quite rarely connected to the Internet; their Internet addresses may change between sessions. Thus within *Campus* we model a two layered network [Fig.3], the outer layer containing all computers which are rarely connected to the network, the inner layer containing computers which are almost permanently connected to the network.
Address changes of computers in the outer layer are not reported to other computers; Address changes in the inner layer are immediately reported to all computers in the inner layer, as well as through their agents to computers in the outer layer as well. Computers in the outer layer send their agents to computers in the inner layer, let them perform their tasks and pick them up again, if necessary via reverse routing. The inner layer provides various service, information and chat 'booths', where agents on behalf of their owners can, e.g.,

- *retrieve actual information from libraries, search machines, faculties' and registrar's blackboards;*
- *exchange information with other agents;*
- *search for individual agents;*
- *co-operate with other agents in setting up individual working groups;*
- *enrol their owners into existing working groups;*
- *arrange dates between their owners;*

Each of these 'booths' contains one context where travelling agents can dock. We are presently developing a

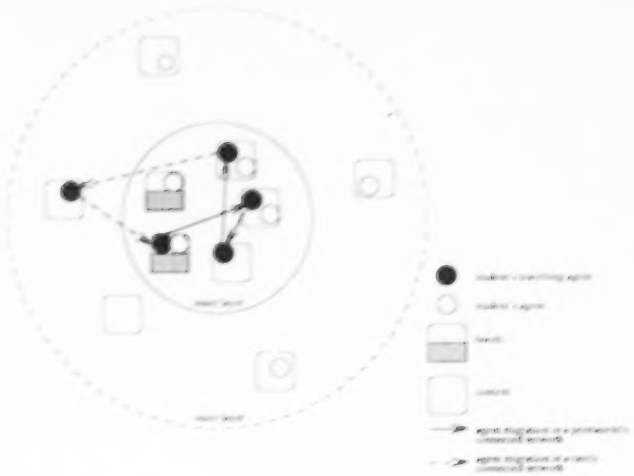

Figure 3: *The two layered network of Campus*

variety of protocols for communication and co-operation between agents and booths.
Campus contains several types of *CIAgents*. They are all derived from the basic agent type of the AAPI package. [Fig.4] depicts the CIAgents' docking- and its route window:

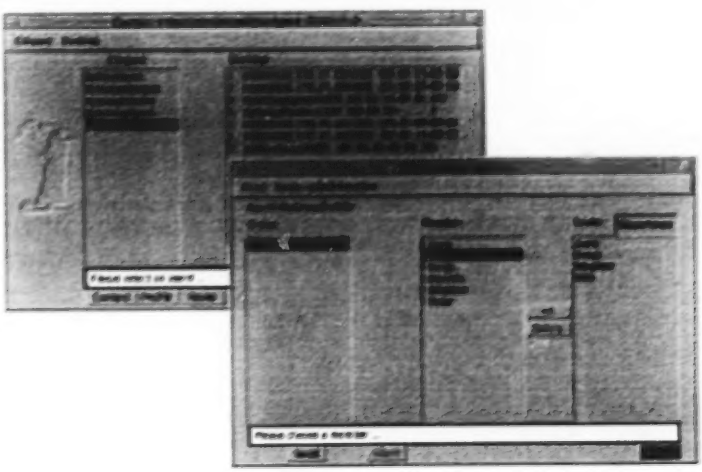

Figure 4: *The CIAgents' docking- and route-window*

The docking window is divided into the boxes *My-CIAgent* and *Docking*. The My-CIAgent box lists the student's personal CIAgents and the docking box gives a view of all the agents that are either working or resting on the student's context. In the *Campus*' route window the student can compose routes for her My-CIAgents, e.g. by selecting members of her learning groups.

5 Conclusion and Future Work

The AAPI package provides a comfortable and easy to use high level interface to implement systems of individual, mobile, autonomous agents in the Internet. It supports decentralised control architectures. With the help of the

BEST COPY AVAILABLE

AAPI package *Campus* is being realised, a platform that, through a variety of different types of agents supports communication and co-operation between a distance teaching university and its students.
Beside the definition of various protocols for inter agent communication and co-operation, security and authentication problems will be our major future concern: runtime environments have to be secured against malicious agents, agents have to be secured against their embedding environments; secure authentication methods have to be developed. Further more, appropriate accounting mechanisms are needed.

6 References

[Gray et al. 96] Gray R., & Kotz D., & Nog S., & Rus D., & Cybenko G. (1996). *Mobile agents for mobile computing*. Department of Computer Science, Dartmouth College, Hannover http://www.cs.dartmouth.edu/reports/abstracts/TR96-285

[Hohl 95] Hohl F. (1995). *Konzeption eines einfachen Agentensystems und Implementation eines Prototyps*. IPVR (Institute for Parallel and Distributed Computer Systems), University of Stuttgart.

[IBM 96] (1996). *Mobile Agents Facility Specification*. International Business Machines Corporation. The Open Group. http://www.trl.ibm.co.jp/aglets/

[Klat 96] Klat P. (1996). *Persistenz als Basis der Migration in einem Mobilen-Agenten-System: Design und Implementierung*. IPVR (Institute for Parallel and Distributed Computer Systems), University of Stuttgart.

[Kotz et al. 96] Kotz D., & Gray R., & Rus D. (1996). *Transportable Agents Support Worldwide Applications*. Department of Computer Science, Dartmouth College, Hannover. http://www.cs.dartmouth.edu/dfk/papers/kotz:
agents.html

[Li et al. 96] Li W., & Mesverschmidt D.G. (1996). *Java-To-Go. Itinaritive Computing Using Java*. University of California at Berkeley. http://ptolemy.eecs.berkeley.edu/dgm/javatools/java-to-go/

[Mira da Silva et al. 97] Mira da Silva M., & Rodrigues da Silva A. (1997). *Insisting on Persistent Mobile Agent Systems*. University of Evora, Portugal, Lisboa.

[Park et al. 97] Park A. S.-B., & Leuker S. (1997). *A Multiple-Agent Architecture Supporting Service Access*. Technische Universitat Aaachen.

[Röhrle et al. 96] Röhrle K., & Hohl F. (1996). *Finden von mobilen Agenten in einem weitverteilten System*. IPVR (Institute for Parallel and Distributed Computer System), University of Stuttgart.

[Straßer et al. 96] Straßer M., & Baumann J., & Hohl F. (1996). *Mole - A Java Based Mobile Agent-System*. IPVR (Institute for Parallel and Distributed Computer Systems), University of Stuttgart, 1996.

[Westhoff 97] Westhoff D. (1997). *AAPI - eine Schnittstelle zur Implementierung mobiler autnomer Agenten*. Germany, Internal technical reference of the FernUniversitat-Gesamthochschule in Hagen.